Experiences with Distributed Objects

Andrew C. Carlson
William R. Brook
Christopher L. F. Haynes

There has been much talk of the potential of object (component-oriented) technology for building distributed systems, especially on-line transaction services, but few opportunities or imperatives to actually use it in production systems. One such opportunity arose in the United Kingdom (UK), when legislation covering the provision of life insurance quotations changed in January 1995, rendering obsolete the existing national quotations service provided by AT&T. The necessity for change, even radical change, in the system that produced these insurance quotations had become clear nine months earlier. At that time, managers and support staff of the existing service became aware that the changes required by the legislation could not be made rapidly enough, nor reliably enough, using conventional development techniques. This paper describes how a team of developers in AT&T ISTEL used distributed objects and the Common Object Request Broker Architecture (CORBA) standard to implement the updated system in time for the change in legislation. Running across more than 50 Windows NT* servers, the system has given distributed objects operational credibility and provided valuable lessons on the technology adoption process.

Introduction

In May 1994 a team of developers at AT&T ISTEL were approached by senior management with a seemingly insurmountable challenge—to implement changes to a legacy insurance quotation service in time to comply with new legislation scheduled to take effect on January 1, 1995. Although senior management considered this a difficult—if not impossible—task, failure to comply would result in a major loss of business.

Figure 1 shows the configuration of the quotations system before it was updated. It ran on a VAX* cluster (a networked group of Digital VAX machines) executing life insurance quotations from a number of third-party insurance companies. Customers, who are financial agents, accessed the system via videotext terminals (a simple color character-based terminal) or Windows* workstations emulating these videotext protocols connected via dial-up serial lines.

Each insurance company specified the algorithms that would be executed to provide the insurance quote for each type of insurance product, such as mortgage life assurance, pensions, or illness. In some cases, they allowed connection to their own computer systems to perform the quotation. To address the changes required to meet the new legislation, each of 49 insurers (other than those whose own computer systems provided quotation services directly to AT&T systems) would be delivering new algorithms for each product, a total of more than 150 algorithms. In all probability, many of these would arrive in November and December of 1994 rather than June or July. With these constraints, the developers

quickly realized they did not have enough time to code and test the new algorithms.

In addition, even though the new legislation resulted in more iterative, CPU-intensive algorithms than those previously in place, the service-level agreements dictated that each quotation issued be completed within 10 seconds. The system was required to support quote comparisons, in which quotations from several companies are provided, for comparison purposes, again within 10 seconds overall.

Proposed Solution

A proposal was made early in the project for insurers to provide a PC application that would perform the calculations required. Because many insurers were already going to undertake this work for their direct sales force, who frequently make home visits and therefore use laptops, this seemed a good suggestion. These applications would then be placed on a server running the MS-DOS* operating system, as shown in Figure 2. The existing Digital VAX/VMS* system would transfer quote service parameters to the relevant machine and receive answers in the form of files from the PC application. Initial proposals were to create a simple file-based mechanism using basic networking software to move the required information between locations.

For commercial reasons, insurers were unwilling to share a machine with other insurers. As a result, three machines were specified for each service—the service machine, running the live services; a normally idle backup machine; and a test machine, used by insurers to test their new variants.

The original proposal, however, had several problems:

- The full system would require as many as 150 PCs running MS-DOS (3 per company), because each insurer would require at least one backup machine and one test machine per product.
- The messaging protocol envisaged would be difficult to keep secure and reliable.
- No support would be available for those insurers who chose to use Windows NT,* Macintosh,* or OS/2* PCs.
- Security and robustness were lacking from the MS-DOS operating system.
- Effective multitasking would be essential to support the interface mechanism being proposed.

Panel 1. Abbreviations, Acronyms, and Terms

API—application programming interface
CASE—computer-aided software engineering
CORBA—Common Object Request Broker Architecture, a standard endorsed by the Object Management Group to handle communication of messages between objects in a distributed, multiplatform environment.
DCE—distributed computing environment
FAT—File Allocation Table, the part of the DOS and OS/2 file system that keeps track of where data are stored on disk.
IDL—interface definition language
LAN—local area network
legacy system—a euphemism for an aging software system with a poor or outdated architecture
OMG—Object Management Group, an international organization that endorses technologies such as open standards for object-oriented applications.
ORB—Object Request Broker
NTFS—NT File System, offered by Windows NT, uses the Unicode character set and allows file names up to 255 characters in length.
object—a self-contained module of data and its associated processing. Objects are software building blocks.
socket—a network communication system
TCP/IP—transmission control protocol/Internet protocol
viewdata—a system for displaying color character screens on low-cost terminals. Developed in the 1970s, it was widely used for travel agencies and financial services. It uses very low data transmission rates and may have only a minimal, telephone-like keypad.
VMS—Virtual Memory System, a multiuser, multitasking, virtual memory operating system for the VAX series, manufactured by Digital Equipment Corporation.
WAN—wide area network
wrapper—a program or process that converts non-object-oriented protocols to object-oriented ones, and vice versa.

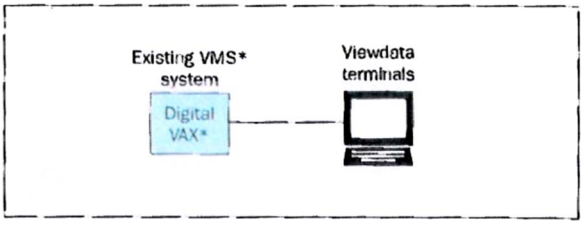

Figure 1. In the original system, the VAX system processed all incoming requests from videotext terminals. The algorithms used to produce the insurance quotes were specific to each company.

- The MS-DOS insurance applications being used are designed to stand alone. The effect of the additional low memory that would be consumed by network drivers and other equipment would have to be evaluated for each application.
- Little or no provision was made for remote management and monitoring in this type of operating environment.

The development team was looking for a communications and operating environment with considerably more functionality and ease of development than a simple file transfer or socket (a network communication system) mechanism could offer. Normally, a full "hands-on" evaluation is conducted on all relevant technologies and products. In this case, there was insufficient time available, and, in fact, many doubted that enough time existed to implement at all. As a result, a paper evaluation of message queuing and file transfer products was carried out. These products were rejected as not providing enough reliability or failure detection.

Because team members had had personal experience with distributed object systems in the past, they looked in the same direction to solve this problem. The proposal was revised to use the Windows NT platform and to adopt the Common Object Request Broker Architecture (CORBA) standard to provide communication facilities (see Figure 3).

Adopting this two-point plan offered a number of advantages:

- Windows NT made it possible to execute DOS, Windows, Windows NT, and OS/2 character applications in a secure environment. Given the multivendor environment, this was critical.
- Windows NT also supported remote administration, an essential feature in such a large number of servers.
- The CORBA distributed object standard and a supporting development tool offered a communications architecture that not only solved the immediate problem in a secure, reliable, and flexible manner, but also provided a long-term strategy for the system by allowing the VAX/VMS systems to be phased out. Architectural improvements were made as well.
- The architecture adopted allowed transactions to be rerouted automatically to alternate server platforms if a hardware fault or software crash occurred. This reduced the overall number of machines required. It also significantly improved on the expected service levels agreed to by insurers for correcting certain types of problems—from several hours down to several seconds. By distributing transactions among all available servers rather than using servers only for contingency purposes, it also provided better performance.
- The Windows NT File System (NTFS) was known to be more reliable and secure than the DOS File Allocation Table (FAT) File System. Because AT&T ISTEL was running this service on behalf of several companies, the confidentiality of such items as commission rates was important.

Distributed Objects

CORBA is an industry standard for how objects communicate with one another across a wide or local area network (WAN or LAN). This is formed around a client/server pairing, in which each object provides services to another. (There are only roles for an object within a system, and, as such, a given object may be either a client, a server, or both at any given instant.) The CORBA standard allows object services to be accessed by name rather than location. This feature enabled the development team to locate available or unloaded servers in preference to specific machines running specific servers. It also balanced the load of transactions among all the available servers and permitted transparent transaction rerouting if a server failed.

The CORBA standard was originally developed in 1991 by the OMG, an industry consortium. Since then it

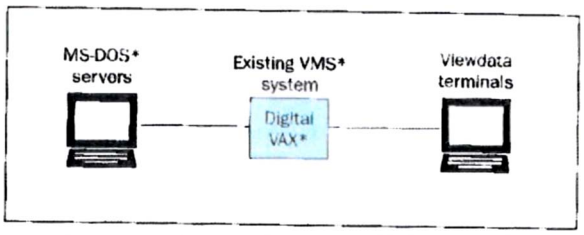

Figure 2. The initial proposal for the updated system shows the VAX system acting as a videotext front-end processor, with DOS machines performing the insurance quotes.

has received support from a wide range of members and vendors, including Digital, IBM, and AT&T. The CORBA 1.2 standard[1] consists of three main features—an interface definition language (IDL), a language mapping, and a statement of system architecture.

First, the CORBA standard specifies an IDL, defining which methods are available on a given object (as shown in Panel 2). It also serves to specify how parameter objects are constituted. Second, a language mapping is defined, specifying which application programming interfaces (APIs) are to be made available by a CORBA-compliant development system, and how the code produced by the IDL compiler will use these APIs. For CORBA 1.2 and its predecessors, only a C language mapping is defined. Last, the standard and its related standards include a statement of the architecture of the system, defining and suggesting how features such as the brokering of services are undertaken. An alternative to IDL, termed *dynamic invocation*, is also defined, which allows interfaces to specific services to be queried at run time. This eliminates the need to compile a client program that has the interfaces and service properties using IDL hard-wired into it.

CORBA 1.2 makes no attempt to standardize on the protocols that are used to transmit objects. It is therefore likely that different vendors' implementations will not be interoperable, unless they have chosen an underlying standard such as the Distributed Computing Environment (DCE) to enable this interoperability.

The CORBA 2.0 standard,[2] formally released in 1995, proposed two significant features: C++ language mapping and an interoperability standard. Extensions are currently being proposed for security and other language mappings such as Smalltalk. These additions to the standard will make possible public, freely accessible services, with generic client programs, perhaps resulting in a free trade in services similar to the model for information access demonstrated by the World Wide Web.

What Is an ORB? The Object Request Broker (ORB), the heart of the CORBA standard, defines the services that are available to the outside world. Every service must create at least one ORB, although there is nothing to stop more than one being created. The ORB is the unit that is brokered to make a given service accessible.

Distributed Objects and the Insurance Problem

The choice of a tool based on the CORBA standard gave the development team several advantages over the original proposal.

Flexibility. One feature of this project was a seemingly endless stream of minor specification changes from the insurance companies. These changes were adopted rapidly, even during the final few weeks of the project. The specifications of the transaction objects were entered into an object-oriented computer-aided software engineering (CASE) tool, Paradigm Plus,* from Protosoft. This tool can generate CORBA IDL and C++ source code and allows customization of generation to support project-specific features. The generation of IDL thus permitted a rapid turnaround on simple specification changes.

Insurance Provider Interface. When the insurance quotation system was completed, each insurance

Panel 2. A sample of CORBA 1.2 IDL

This sample of CORBA 1.2 IDL shows how an insurance product and its interface are defined, in this case for an endowment mortgage.

```
interface EndowmentMortgageRequest;
interface EndowmentIllustration;
interface EndowmentMortgageRequestHnd;
interface EndowmentIllustrationHnd;

interface EndowmentMortgage : Mortgage
{
void illustrate(in EndowmentMortgageRequest rqst,
  inout EndowmentIllustration     illust);

} ;
```

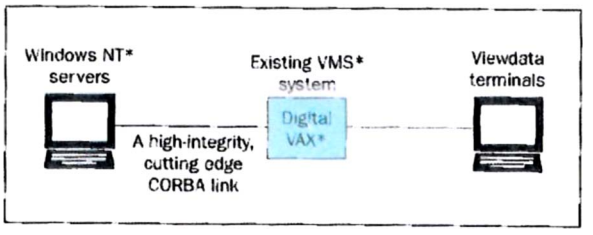

Figure 3. In the final proposal, the VAX system continues to process incoming videotext information, but the insurer's transactions are now processed by Windows NT servers using a CORBA transport.

company would provide its own executable program for issuing quotations. In theory, the industry consortium was imposing standards on the route to be adopted; in practice, the chosen solution involves the adaptation of a wide variety of quotation applications to communicate with the rest of the system in a standard manner. Given that virtually every application represented a unique combination of development language (hence, level of I/O support) and operating system, there was really only one option—a "low-tech" approach. The team developed a CORBA-aware wrapper running in Windows NT native mode, which would receive incoming insurance quotation requests. (A *wrapper* is a program or object that makes another program appear to the object environment. In this case, CORBA was a native object, even though it knew nothing of the object or CORBA.) The information contained in each request was then written to a specified file for consumption by the insurance quotation engine, which executed it in a variety of modes, but primarily in the MS-DOS emulation mode. The wrapper, on the other hand, always ran in native Windows NT mode. When the quotation was completed, the insurer application wrote an output file, which the wrapper detected and read from to create the reply transaction. This was returned in the form of a CORBA reply.

While the file-based interface approach satisfies many of the insurance providers' needs, it also creates some problems:

- All files must be named in a way that avoids clashes. The wrapper, termed the "server proxy," handles all file name management. When it starts up, it tells the insurance provider's application which names it should use.
- The server proxy cleans up the old files, although a well-behaved insurance provider's application is expected to delete its input file after it has been processed.
- Synchronization between reader and writer is difficult to achieve, given the number of file and other input/output implementations involved. This was solved by renaming files once writing was complete and before reading could begin.
- The use of files implies that the reader must poll the disk to see if the file exists. Because Windows NT increases task priority based on I/O frequency, applications were gaining too much priority when they were polling (more so than when performing a quotation). This was resolved by using the server proxy to lower the task priority of the insurer's application when it was not processing a quotation.

Architectural Issues

Because of the time constraints on the quotation system's initial delivery, the VAX/VMS system was retained to support the existing user interface, some minor corralling and processing, and the hardware for dialing into the system. By adopting the CORBA standard, however, the development team retained the option to offer Windows-based client programs directly to the customer at some later date. This would bypass the existing VAX/VMS system altogether, allowing the team to gradually move customers to a Windows workstation/CORBA-based system, while using the same system to support the original videotext users. In short, the choice met both their pressing short-term goals, and also their long-term, strategic aims for reengineering this legacy service. The CORBA architecture, chosen as the transport mechanism, was approved by the customers—the insurers—who were happy to see a standards-based approach used in this mission-critical project.

Choice of Distribution Technology. The main constraint on this decision was the need to find a mechanism that would support both a relatively old operating system and a new one, namely: VMS and Windows NT. Unfortunately, at that time most of the marketplace supported only one or neither.

The development team was looking for a product that offered considerably more in functionality and ease of development than a simple "data pipe" such as sockets.

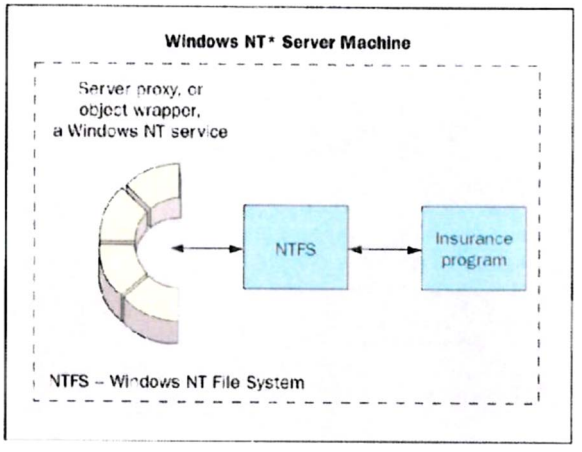

Figure 4. The Windows NT server, and how it relates to a DOS insurance application. The application service, which provides quotations, is made up of two components that exchange information via the file system. To the left of the file system is the wrapper, or proxy, which looks like an object to the CORBA system. "Inside it," metaphorically speaking, is the DOS application provided by the insurance company. The shape is half of a symbol typically used to denote an object, representing the fact that the wrapper has an object side and a non-object side.

They were pleasantly surprised to find a distributed object product loosely based on the OMG CORBA standard, namely DOME. Developed by Object Oriented Technology (which has since ceased trading), it supported the required platforms, and its supplier was only 25 miles from the development team's office. DOME fully supported the CORBA IDL, but it was based on the C++ language, rather than C. At that time, the C language was the only specified language mapping for CORBA (CORBA 2.0 now includes a full C++ mapping). Having used C++ or Smalltalk for development for a number of years, the team was pleased to be able to use this language rather than the standard CORBA 1.2 C language mapping.

From previous experience with projects involving intermachine communication, the developers expected that it would take at least several weeks before they could demonstrate a minimal form of communication. In fact, they had a working demonstration between two Windows NT machines after only two days. The use of IDL was a significant factor in the ease with which they were able to achieve this result. IDL provides an object developer with a very simple, natural way of describing a distributed system. This description is then put through a compiler to provide the communication capability of the finished system.

This process was made easier by using a generating CASE tool, Paradigm Plus, to generate the required IDL. All specifications for the system were retained in the CASE tool, and changes were made to a single design repository. The development team could then generate IDL or C++ as required, and/or use the IDL to generate C++ interfaces.

The IDL is used to generate "stub" code, which is built in to all components of the distributed system. At present, it is necessary to ensure that all parts of the system are generated from a single IDL source (and the same version of that source). This may pose problems in a system with many machines, particularly if frequent changes are expected, because any change will involve updating all machines at the same time.

An alternative to the IDL approach involves having the components of the distributed system declare themselves to each other (at run time, for example) via the CORBA Dynamic Invocation Interface. This type of architecture makes it possible to cope with changes in a more enlightened fashion. It also promises new possibilities, such as the use of higher-level development tools (for example, fourth-generation languages, or 4GLs), Microsoft Visual Basic, and others, to integrate with the distributed system. This possibility has not yet been investigated.

Final Architecture

The existing viewdata software is retained to continue support for this method of workstation access, provide user authorization, and coordinate all system activities performed on behalf of the user. The viewdata software makes calls via a simple function call interface into the client side of the C++ Object Model Implementation, which defers most operations to a server via the CORBA DOME system. DOME communicates from the VAX to the Windows NT PCs through sockets using the TCP/IP network protocol. A custom server application

(the server proxy) is written as a Windows NT system service and performs the following roles:

- Wrappering of the file-based interface mechanism to make it appear as an object,
- Buffering of quotation requests to ensure that the insurance provider application is only given one request at a time,
- Error handling and cleanup for the file-based interface mechanism, and
- Startup/shutdown processing and control of the process priority of the insurance provider's application.

Because the server proxy is a Windows NT service, it restarts automatically should a machine failure occur, thus ensuring that it is on line again as soon as possible. In the meantime, the failure is detected and other servers take up the load. Structured exception handling ensures that even a catastrophic failure of the software still results in a correct shutdown of the insurance program, a DOS application. On one occasion, however, this resulted in a graceful cleanup after a fatal application error, which made detection of the error very difficult, despite error logging.

DOME also provides the Location Broker, a distributed directory service with which services register when starting up. It is then queried by the VAX to locate a server capable of performing the quotation required. For commercial reasons and clarity of resource allocation, each insurer is assigned specific NT server machines on which to run its software, which may consist of several insurance products.

Extensions to DOME. To make the DOME system effective in the context of this project, it had to rapidly detect errors—either communications or otherwise—and reroute transactions to an alternate service. At the project start, the DOME system relied on TCP/IP for a non-adjustable 90-second time-out. It was not possible to detect whether a service or communication link had failed within this time. With assistance from Object Oriented Technology, the vendors of DOME, the development team took two actions:

- A configurable time-out was incorporated and reduced to 5 seconds for this project.
- The location broker regularly checked the integrity of each server by using the low-level "ping" facility (a TCP/IP sanity checking feature).

Using a second ORB to check service integrity imposed performance overheads and actually reduced reliability. This method was abandoned.

Results and Lessons Learned

The development team learned some key lessons that can be applied to reviving systems based on legacy systems in a reasonable time frame using object technology. The development team believes that the use of objects and the novel training methods used to teach object technology were important factors in the success of this project.

Legacy Systems. This project has demonstrated that innovative technology can be used to give new life to old systems that might otherwise be unable to fulfill the expectations of today's systems. Alone, the VAX-based system was incapable of meeting the new requirements. Introducing distributed object technology enabled the team to migrate the resource-intensive functionality to a more capable platform without requiring migration and consequent rewriting of the remaining system. The effort involved in these tasks would make the project totally unfeasible in the time frame required. This is a key lesson for technology providers: Evaluate new technologies and recognize the possibility that one of them may be useful in reviving systems based on older technology. If the older hardware and operating systems are supported or can be interfaced to, even if only at a reduced level of functionality, it may be worth the effort.

Object Training. It is a common maxim, recently stated by Marie Lenzi in *Object Magazine*,[3] that one should keep object pilots simple, limit the number of new technologies, and not use them for mission-critical and time-critical projects. The development team wholeheartedly supports these goals.

Unfortunately, real life is seldom so kind. In practice, senior management is usually only willing to consider untried methods such as object orientation and distribution when backed into a corner—in short, when they are faced with time-critical, mission-critical problems, and the risk of the new is less than the risk of conventional tools, which are certain to fail. Such was the case with this project.

Despite these risks, the team managed because of two factors. First, two team members who were very experienced with objects were able to guide the

remaining members of the team. Second, the team used novel methods of training.

On several occasions the team was advised to adopt objects in a "slow, but sure" way, for example, teaching a new feature of C++ to existing C programmers each week. The team felt that this was the wrong way. Object orientation involves a completely different mindset than conventional program development. Rather than taking a "softly-softly" approach, they had to "go for broke," breaking people's existing mindset before introducing the new one. With this in mind, all of the new staff were sent to a one-week Smalltalk course (despite the intention to use C++ on this project). This forced the staff members to come to terms with objects, rather than muddling through using elements of C in a C++ program. It took about two months for the programmers to thoroughly grasp object orientation; during this time, the project was largely unproductive. When the training period concluded, however, the project began to catch up with its deadlines rapidly, eventually meeting all its goals, plus some additional ones that arose along the way.

Since then the business unit has extended this training to introduce all of the appropriate staff, including analysts, to object orientation. These are some of the basic maxims that they follow:

- Formal training can give a developer a basic grounding in C++ and OO techniques, but there will still be a long way to go at the end of the course.
- The use of experienced personal mentors has proven beneficial for technology transfer in both C++ development and object-oriented techniques.

Reuse. Software components are often constructed in a generic manner so they can be used in other, possibly unforeseen, applications. Just such an approach has been applied on this project. A second form of reuse, namely, buying commercially available components, is easily overlooked, even though it is of more immediate benefit. This involves the careful selection of cooperating class libraries purchased from third parties. Further details are beyond the scope of this paper.

Reusable Components. If the additional effort required to create reusable components is to bear fruit, the organization must provide an environment that fosters reuse, both by developing and managing reusable components and by encouraging their reuse.

One effective approach is to promote awareness of the availability of components within the development community. The precise nature of such a mechanism depends on the size and culture of the organization. The business unit in question has adopted a regular forum of key developers from around the company, at which such matters are discussed.

Documentation or training materials should be written with the reuser in mind, rather than being an exposition on how the components themselves are designed. Such materials should contain examples showing how the components can be applied.

Effective ownership of corporate software assets should be assigned to appropriate work groups, including support for the reusers, review of new components, management of changes, and contribution to the documentation and testing of components to the level required for reuse. Where development focuses on research or infrastructure, reuse should be easy to achieve. If the organization is totally project focused, as is the case with AT&T ISTEL, it may be more difficult to introduce a reuse culture.

Distributed Systems. On this project, the use of distributed objects and the CORBA standard taught the development team many lessons, one of which is this: Select the CORBA toolset very carefully. Although the developers made the only choice available at the time, they had to spend a significant amount of time improving the integrity checking of the system. They had to enable the system to detect almost immediately when a server had failed so transactions could be rerouted to an alternate service. If a project requires integrity checking, its developers should select a more capable implementation of CORBA, rather than writing extensive integrity checking code, which is a nontrivial matter.

Conclusion

This project was very challenging and at times stressful for all involved in its implementation. Looking back, the development team feels justified in saying that object-oriented development has proven its ability to deliver a system that meets its customers' needs. The quotation server environment was developed throughout using object-oriented techniques. The system was delivered and working on time at the beginning of 1995.

Even when it encountered excessive loads owing to a post-holiday backlog within its first five hours of operation, the system exceeded its design load without failure. It is continuing as of this date to provide a high availability service to AT&T's insurance customers in the UK.

Object orientation is now being used on numerous projects within both financial and nonfinancial market arenas. If the development team had to propose some unbreakable rules regarding the introduction of objects, they would recommend two things:

- Use experienced staff to ensure success—object orientation is learned from experience, not from courses. Borrow or buy in experience if you have none in your business unit.
- Be radical. Adopt object orientation wholeheartedly or not at all, and adopt it quickly in one small group rather than by a gradual transition in many groups.

***Trademarks**

Macintosh is a registered trademark of Apple Computer, Inc.

MS-DOS, Windows, and Windows NT are trademarks and Visual Basic is a registered trademark of Microsoft Corporation.

OS/2 is a registered trademark of International Business Machines Corporation.

Paradigm Plus is a trademark of Protosoft.

VAX, VMS, and Digital are registered trademarks of Digital Equipment Corporation.

References

1. *The CORBA 1.2 Standard*, The Object Management Group, 1993.
2. *The CORBA 2.0 Standard*, The Object Management Group, 1995.
3. Marie A. Lenzi, "Successful takeoff or crash and burn?," *Object Magazine*, Vol. 3, No. 2, July/August 1993, p. 4.

Useful Web Sites

Object Management Group: www.omg.org, ftp.omg.org

AT&T ISTEL: www.attiaa.attistel.co.uk:8000 (accessible also from the www.att.com index)

(Manuscript approved March 1996)

Andrew C. Carlson *is a senior member of technical staff in the Advanced Services group of AT&T Value Added Services, Europe, Middle East and Africa (EMEA), in Redditch, United Kingdom. While this project was being conducted, he worked as a consultant for AT&T ISTEL Finance, Travel and Retail Services, now a division of AT&T Value Added Services UK, which runs the service. His current responsibilities are to offer consultancy and team leading skills in object orientation, the Internet, and distributed systems to the Value Added Services business units across the EMEA region. Mr. Carlson joined AT&T in 1986, after receiving a B.Sc. degree in engineering science and technology from Loughborough University of Technology, England.*

William R. Brook *is a senior member of technical staff and a project manager with the Advanced Services group of AT&T Value Added Services, EMEA, in Redditch, United Kingdom. During this project, he worked for AT&T ISTEL European Technology Group, charged with the task of introducing object orientation to AT&T ISTEL. His current responsibilities are to offer consultancy and project management skills relating to object orientation distribution, and the Internet. He is now managing a project that reviews billing and customer care technologies, primarily relating to the Internet and WorldNet, for Value Added Services EMEA. Mr. Brook joined AT&T in 1987, after receiving a B.Sc. degree in physics from the University of Birmingham, England. He is currently undertaking a B.A. degree in history from the Open University, England.*

Christopher L. F. Haynes *is director of Advanced Services for AT&T Value Added Services, EMEA, in Redditch, United Kingdom. During this project, he was the chief technical officer for AT&T ISTEL and manager of the European Technology Group, and was instrumental in the use of objects in this project. Mr. Haynes is currently responsible for content-related research and development for all new AT&T on-line and associated services in the EMEA region. He received a B.Sc. degree in electronic and electrical engineering from the University of Birmingham, England, and an M.Sc. degree in radar and radio communication from the University of Birmingham. He also held the post of lecturer in computing at the University of Surrey in the Electrical Engineering Department. Mr. Haynes, who holds three patents in radar and signal processing and*

currently teaches information technology strategy to the M.Sc. business course at the University of Warwick, England. joined AT&T in 1991.